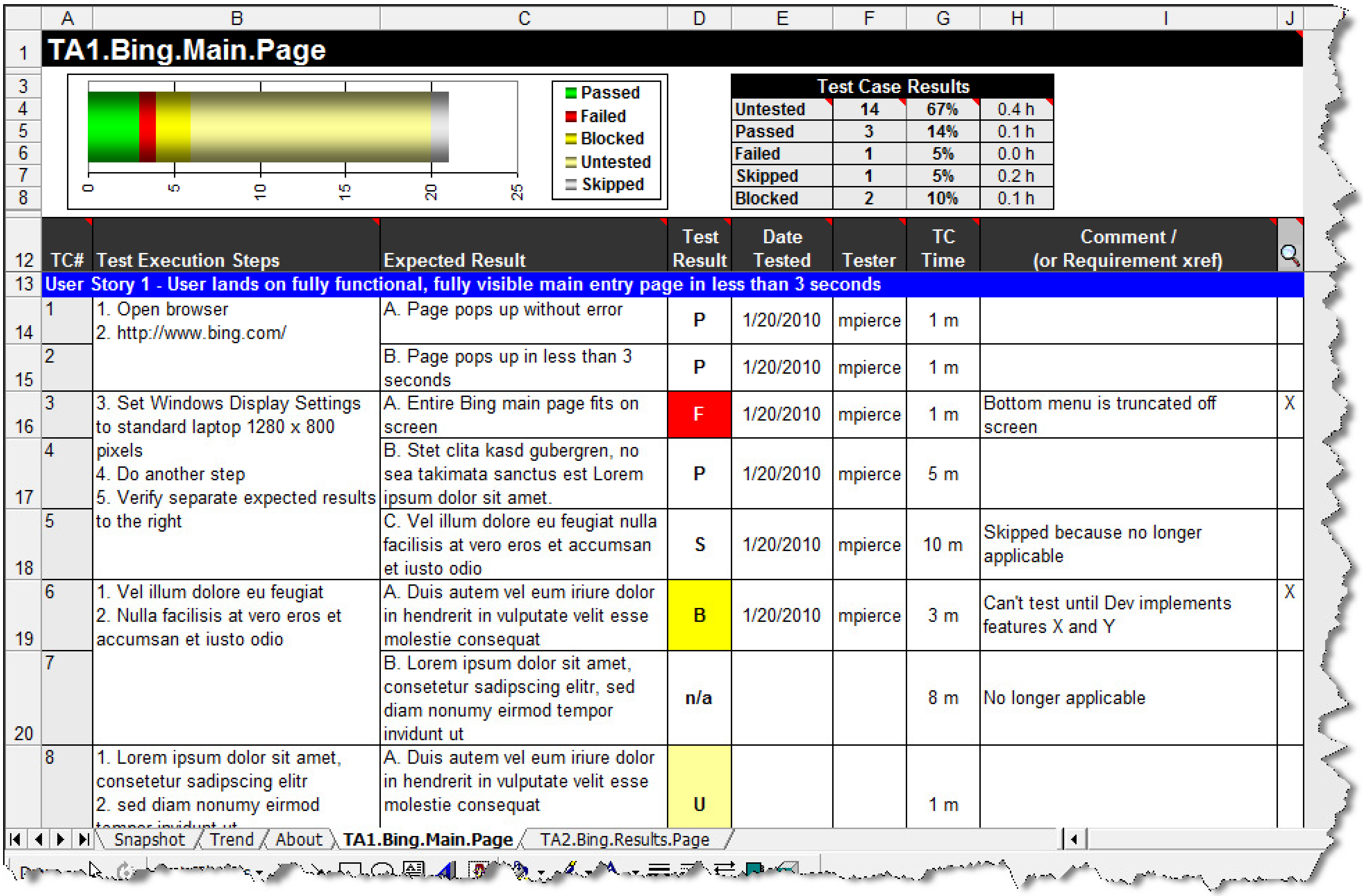
Ce conține un caz de test (Test Case)?

Regression

Id, Description, Feature, Info, Revision, Steps
[Apache OO]

Id, Title, Preconditions, Steps, Expected Results, Status, Tested product/component/method, Link to specs,...

Test Case. Exemple



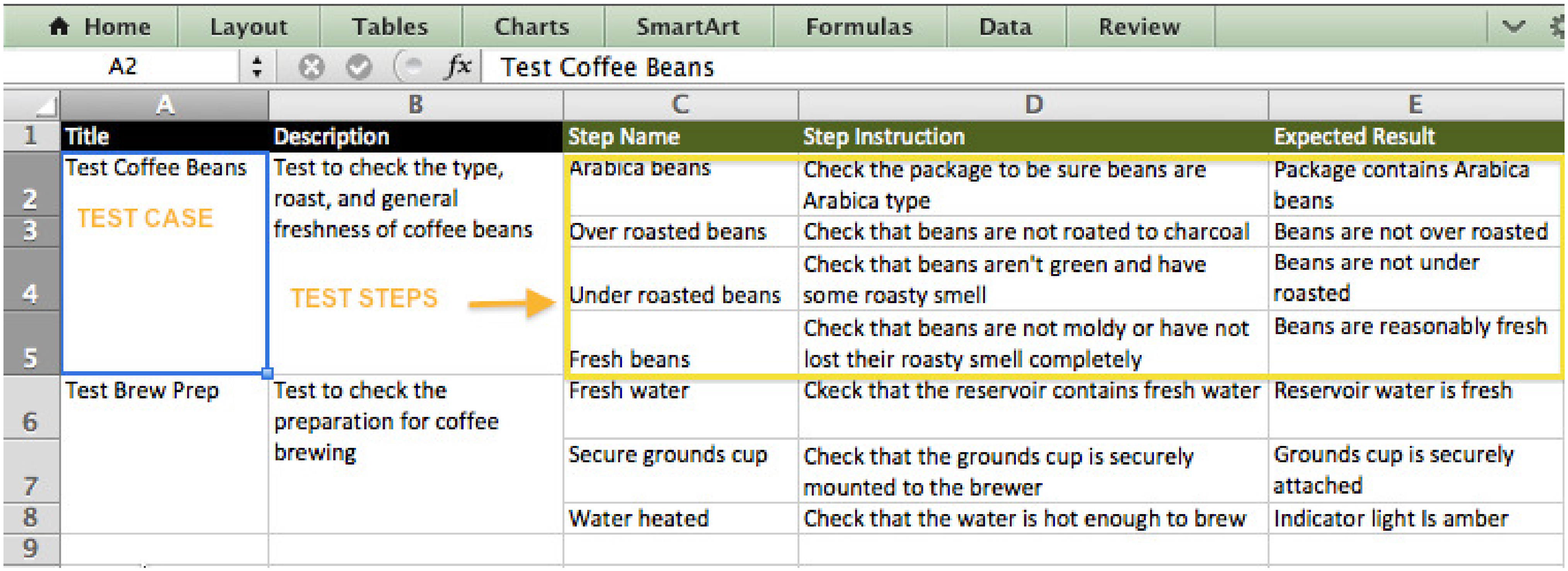
TA1.Bing.Main.Page

🟩 Passed
🟥 Failed
🟨 Blocked
🟨 Untested
⬜ Skipped

Test Case Results			
Untested	14	67%	0.4 h
Passed	3	14%	0.1 h
Failed	1	5%	0.0 h
Skipped	1	5%	0.2 h
Blocked	2	10%	0.1 h

TC#	Test Execution Steps	Expected Result	Test Result	Date Tested	Tester	TC Time	Comment / (or Requirement xref)	
User Story 1 - User lands on fully functional, fully visible main entry page in less than 3 seconds								
1	1. Open browser 2. http://www.bing.com/	A. Page pops up without error	P	1/20/2010	mpierce	1 m		
2		B. Page pops up in less than 3 seconds	P	1/20/2010	mpierce	1 m		
3	3. Set Windows Display Settings to standard laptop 1280 x 800 pixels 4. Do another step 5. Verify separate expected results to the right	A. Entire Bing main page fits on screen	F	1/20/2010	mpierce	1 m	Bottom menu is truncated off screen	X
4		B. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.	P	1/20/2010	mpierce	5 m		
5		C. Vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio	S	1/20/2010	mpierce	10 m	Skipped because no longer applicable	
6	1. Vel illum dolore eu feugiat 2. Nulla facilisis at vero eros et accumsan et iusto odio	A. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat	B	1/20/2010	mpierce	3 m	Can't test until Dev implements features X and Y	X
7		B. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut	n/a			8 m	No longer applicable	
8	1. Lorem ipsum dolor sit amet, consetetur sadipscing elitr 2. sed diam nonumy eirmod	A. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat	U			1 m		

Test Case. Exemple

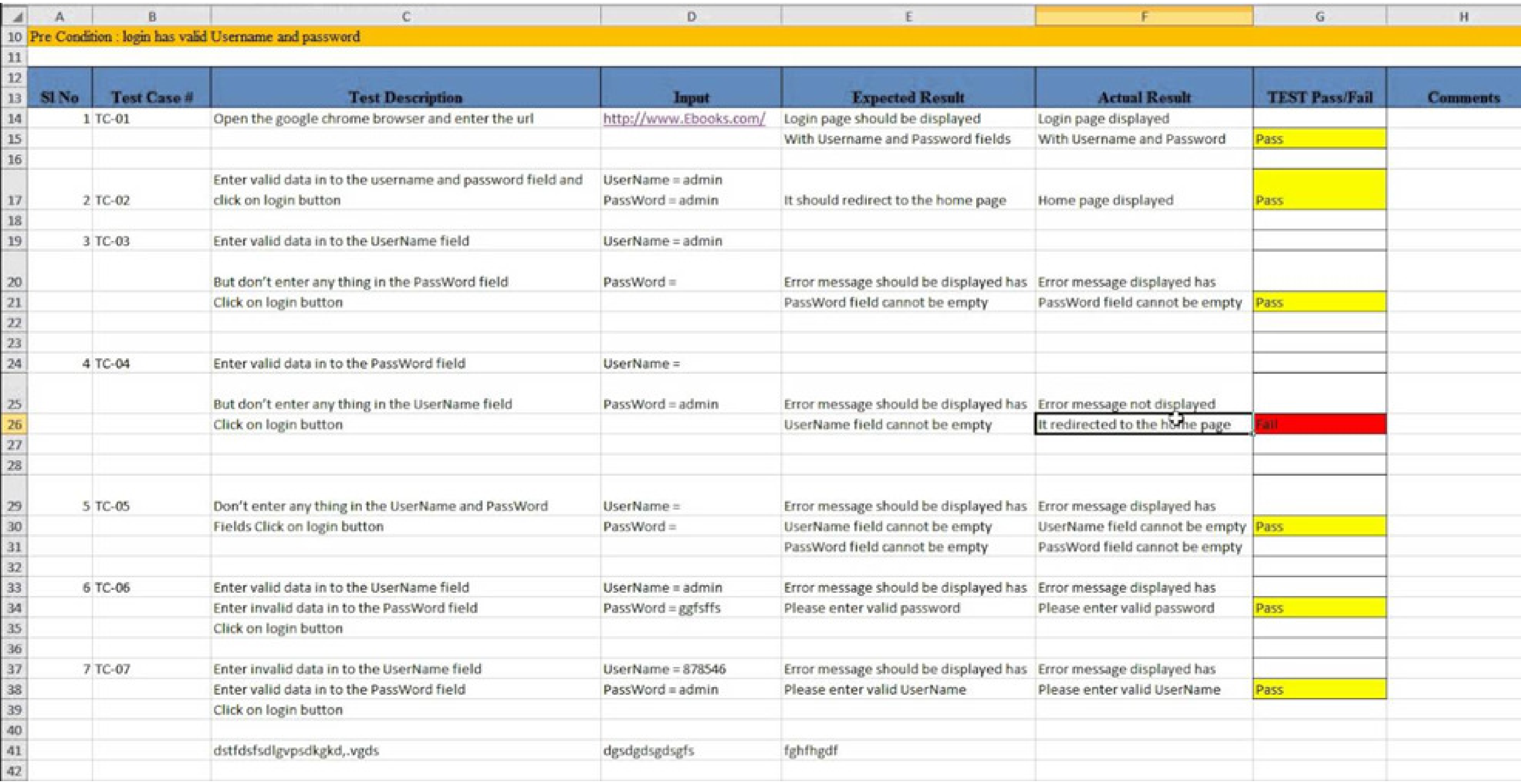


	A	B	C	D	E
1	Title	Description	Step Name	Step Instruction	Expected Result
2	Test Coffee Beans	Test to check the type, roast, and general freshness of coffee beans	Arabica beans	Check the package to be sure beans are Arabica type	Package contains Arabica beans
3			Over roasted beans	Check that beans are not roated to charcoal	Beans are not over roasted
4			Under roasted beans	Check that beans aren't green and have some roasty smell	Beans are not under roasted
5			Fresh beans	Check that beans are not moldy or have not lost their roasty smell completely	Beans are reasonably fresh
6	Test Brew Prep	Test to check the preparation for coffee brewing	Fresh water	Ckeck that the reservoir contains fresh water	Reservoir water is fresh
7			Secure grounds cup	Check that the grounds cup is securely mounted to the brewer	Grounds cup is securely attached
8			Water heated	Check that the water is hot enough to brew	Indicator light Is amber
9					

Test Case. Exemple

Test Case ID	Test Case Description	Dependencies	TC Ready for Review	Functional C1	Functional C2	SIT C1	Performance	Requirements ID
TC1.0 – New Contact	Create new contacts: Login → Contacts →*New Contacts* → *Verify Data Elements*	N/A	Approved	X				34, 112, 212, 243, 244
TC2.0 – New Contract Standard	Validates the creation of a new contract and the approval process: Login → Contract Holder → *New Contract Holder* → Contracts → *New Contract* → Commissions → *Commissions* → Logout	TC1.0, TC1.1, Username, Password, Appropriate	Approved	X			X	24, 25, 36, 37, 40, 44, 59, 99, 107, 194, 196, 226, 233, 235, 244, 254, 256
TC 2.3 - Test a	Testing a Campaign: Login -> *Login* -> Program Plans -> *Query Program Plan* ->	TC1.0, TC1.1,	Not Reviewed	X		X		129, 150, 153,154, 159,
TC2.0 – New Contract Standard	Validates the creation of a new contract and the approval process: Login → Contract Holder → *New Contract Holder* → Contracts → *New Contract* → Commissions → *Commissions* → *Approval process* → *Logout*	TC1.x, LOVs and State model	Not Reviewed		X	X		24, 44, 59, 77, 79, 85, 98, 99, 100, 112, 133, 135, 138, 193
TC3.0 – Contracts	Validates that this process will fail correctly (negative test): Login →New Contract Holder → *New Contract Holder* → Contracts → *New Contract* → Commissions → *Commissions* Approval Process → *Rejection* → *Resubmit* → *Logout*	TC1.x, LOVs and State model	Not Reviewed		X	X		24, 44, 59, 77, 79, 85, 98, 99, 100, 107, 112, 133, 135, 138, 193,
TC4.3 – Contracts	Validate the converted data in the Contracts fields are correct based on test inputs (using field names from the Design document).	TC1.x, TC2.x, Contracts data	Not Reviewed			X		51, 112, 143 198, 200, 201, 204, 265

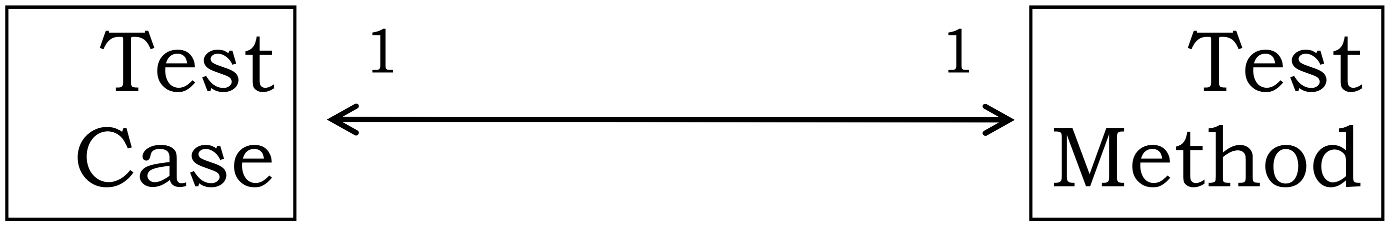
Test Case. Exemple

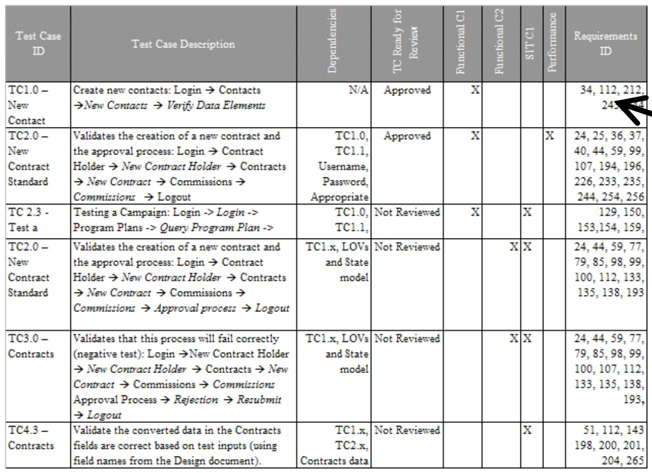


Pre Condition : login has valid Username and password

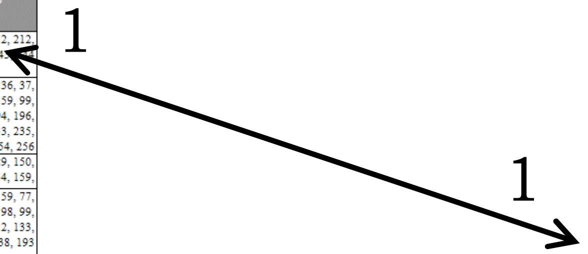
Sl No	Test Case #	Test Description	Input	Expected Result	Actual Result	TEST Pass/Fail	Comments
1	TC-01	Open the google chrome browser and enter the url	http://www.Ebooks.com/	Login page should be displayed	Login page displayed		
				With Username and Password fields	With Username and Password	Pass	
		Enter valid data in to the username and password field and	UserName = admin				
2	TC-02	click on login button	PassWord = admin	It should redirect to the home page	Home page displayed	Pass	
3	TC-03	Enter valid data in to the UserName field	UserName = admin				
		But don't enter any thing in the PassWord field	PassWord =	Error message should be displayed has	Error message displayed has		
		Click on login button		PassWord field cannot be empty	PassWord field cannot be empty	Pass	
4	TC-04	Enter valid data in to the PassWord field	UserName =				
		But don't enter any thing in the UserName field	PassWord = admin	Error message should be displayed has	Error message not displayed		
		Click on login button		UserName field cannot be empty	It redirected to the home page	Fail	
5	TC-05	Don't enter any thing in the UserName and PassWord	UserName =	Error message should be displayed has	Error message displayed has		
		Fields Click on login button	PassWord =	UserName field cannot be empty	UserName field cannot be empty	Pass	
				PassWord field cannot be empty	PassWord field cannot be empty		
6	TC-06	Enter valid data in to the UserName field	UserName = admin	Error message should be displayed has	Error message displayed has		
		Enter invalid data in to the PassWord field	PassWord = ggfsffs	Please enter valid password	Please enter valid password	Pass	
		Click on login button					
7	TC-07	Enter invalid data in to the UserName field	UserName = 878546	Error message should be displayed has	Error message displayed has		
		Enter valid data in to the PassWord field	PassWord = admin	Please enter valid UserName	Please enter valid UserName	Pass	
		Click on login button					
		dstfdsfsdlgvpsdkgkd,.vgds	dgsdgdsgdsgfs	fghfhgdf			

xUnit. 1 Test Case = 1 Test Method





Test Case ID	Test Case Description	Dependencies	TC Ready for Review	Functional C1	Functional C2	SIT C1	Performance	Requirements ID
TC1.0 – New Contact	Create new contacts: Login → Contacts →*New Contacts* → *Verify Data Elements*	N/A	Approved	X				34, 112, 212, 24[illegible]
TC2.0 – New Contract Standard	Validates the creation of a new contract and the approval process: Login → Contract Holder → *New Contract Holder* → Contracts → *New Contract* → Commissions → *Commissions* → Logout	TC1.0, TC1.1, Username, Password, Appropriate	Approved	X			X	24, 25, 36, 37, 40, 44, 59, 99, 107, 194, 196, 226, 233, 235, 244, 254, 256
TC 2.3 - Test a	Testing a Campaign: Login *-> Login ->* Program Plans -> *Query Program Plan ->*	TC1.0, TC1.1,	Not Reviewed	X		X		129, 150, 153,154, 159,
TC2.0 – New Contract Standard	Validates the creation of a new contract and the approval process: Login → Contract Holder → *New Contract Holder* → Contracts → *New Contract* → Commissions → *Commissions* → *Approval process* → *Logout*	TC1.x, LOVs and State model	Not Reviewed		X	X		24, 44, 59, 77, 79, 85, 98, 99, 100, 112, 133, 135, 138, 193
TC3.0 – Contracts	Validates that this process will fail correctly (negative test): Login →New Contract Holder → *New Contract Holder* → Contracts → *New Contract* → Commissions → *Commissions* Approval Process → *Rejection* → *Resubmit* → *Logout*	TC1.x, LOVs and State model	Not Reviewed		X	X		24, 44, 59, 77, 79, 85, 98, 99, 100, 107, 112, 133, 135, 138, 193,
TC4.3 – Contracts	Validate the converted data in the Contracts fields are correct based on test inputs (using field names from the Design document).	TC1.x, TC2.x, Contracts data	Not Reviewed			X		51, 112, 143 198, 200, 201, 204, 265



```
[Test]
public void TestMethod(){
    //arrange
    …
    //act
    …
    //assert
    …
}
```

Test Methods naming

- No standards !
- Self-explanatory
- Easy to read/understand

Test Method name. Good to have:

- Unit of work/SUT (method/class)
- State under test (short and meaningful description of the test scope)
- Expected behavior

```
Public void Sum_NegativeNumberAs1stParam_ExceptionThrown()

Public void Sum_NegativeNumberAs2ndParam_ExceptionThrown ()

Public void Sum_simpleValues_Calculated ()
```

Stunt Doubles

Stunt Doubles

Test Doubles

[Gerard Meszaros]

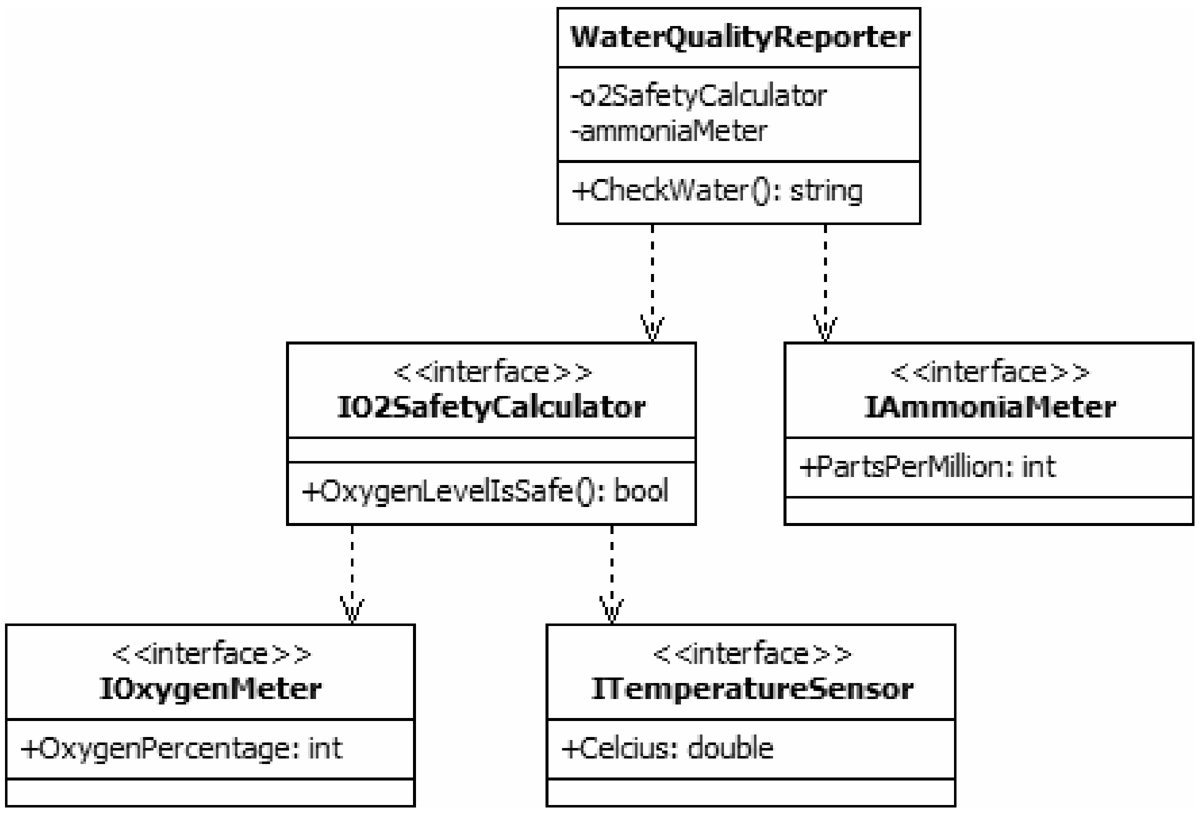
What if the SUT was **designed/not designed** that it could be **tested in isolation** of other pieces of software?

- If testing in isolation is possible -> easy
- If testing in isolation is **not** possible -> 2 options:
  - testing the SUT together with all the dependencies
  - try to **isolate** the SUT **with** various **techniques**

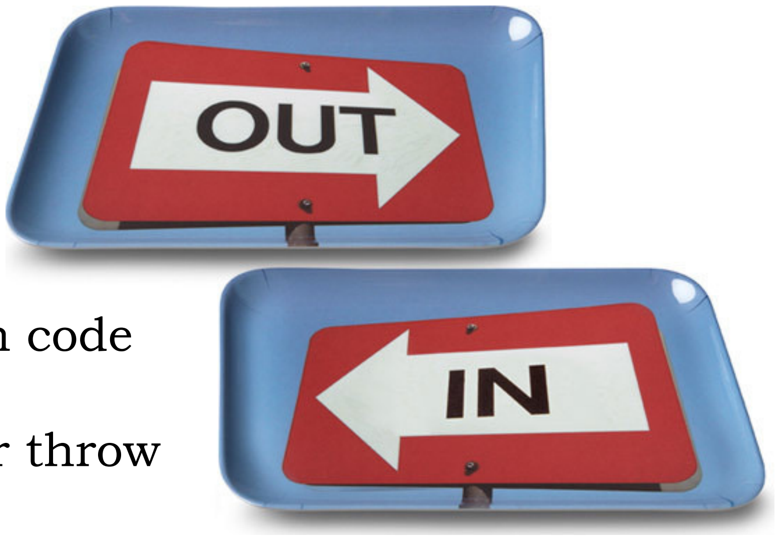
Test Doubles. Example

blackwasp.co.uk

Testing the quality of water in an aquarium. IAmmoniaMeter, IOxygenMeter and ITemperatureSensor interfaces are implemented by classes that control specialist hardware with sensors that are placed into the water being checked. We need to test the WaterQualityReporter class. **HOW?**



Test Doubles. Indirect Inputs/Outputs



- testing classes in groups

  -> **very hard** to cover **all the paths** through code
- **DOC** (depended-on components) return values or throw exceptions to the SUT (the SUT depends on the DOC). Sometimes impossible to make the DOC respond as needed by the SUT
- **Indirect Inputs** received from the DOC could be **unpredictable** (temperature, oxygen level, system clock, calendar,..) or DOC could be **not implemented yet**
- monitoring the side effects of exercising the SUT difficult to monitor -> **Indirect Outputs**

Test Doubles. Indirect Inputs

- untested SUT paths = untested code
- challenge to test all paths
- testing in production = probably catastrophic failures
- **the solution: make** the DOC return values/throw exceptions such as the SUT paths should be exercised regarding DOC **mocked** (simulated/artificial/inauthentic) **behavior**

Test Doubles. Indirect Inputs

- for testing the SUT with **Indirect Inputs** we need to control the DOC, we need to cause the DOC to *return every possible value* and to *throw every possible exception* in the context of given requirements

- can not use the real component (DOC) for producing needed **II**:
  - Real component can not be manipulated
  - Real component can be manipulated but is cost-effective
  - Real component is not available/not yet implemented
- **solution**: implement a Test Double that will simulate (hardcoded or not) DOC behavior

Test Doubles. Indirect Outputs

- **Indirect Inputs** = return values/thrown exceptions by DOC

- What if DOC return/throws nothing? How can we test SUT behavior?

- Example: DOC = a message logging system. No returns to the SUT

- How can we test that the SUT called correctly the DOC? (Or how can we test the **Indirect Outputs**?)

- Not testing Indirect Outputs leads to **Untested Requirements**

Test Doubles. Indirect Outputs

- to test **Indirect Outputs** we must be able to observe **SUT calls to DOC**
- Back Door Verification
- we need a **Test Spy** to record SUT calls to DOC
- make assertions on the expected calls and the recorded calls

Test Doubles. Types

- **Dummy Object** - **II** (Indirect Inputs)
- **Test Stub & Test Spy** - **II & IO** (Indirect Outputs)
- **Mock Object** - **IO**
- **Fake Object** - alternative implementation of the same functionality

Test Doubles. Dummy Obj

- never used in the SUT business logic
- irrelevant behavior
- just needed by the SUT as a parameter

```
[Test]
public void TestMethod(){
    //arrange
    var product = new Product("Dummy name", getUID());
    var invoice = new Invoice (new DummyCustomer());

    //act
    invoice.AddItemQuantity(product, 1);

    //assert
    Assert.AreEqual(invoice.GetLineItems().size(),1);
}
```

Test Doubles. Test Stub

- an object that delivers Indirect Inputs to the SUT
- allows to exercise untested path through the SUT
- *Responder Test Stub* injects valid/invalid Indirect Inputs into the SUT

through returns from method calls

- *Saboteur Test Stub* raises and injects exceptions or errors into the SUT

STATE verification – verify if SUT worked correctly by examining the **state** of the SUT after being exercised

Test Doubles. Test Stub. Example

```
interface ISecondDeep { Boolean SomethingToDo(String str); }

class SecondDeep : ISecondDeep { ... }

class FirstDeep {
    readonly ISecondDeep secondDeep;
    public FirstDeep(ISecondDeep secondDeep) { this.secondDeep = secondDeep; }
    public String AddA(String str) {
        var flag = this.secondDeep.SomethingToDo(str); ...
    }
}
```

```
class SecondDeepStub : ISecondDeep {
    public Boolean SomethingToDo(String str) { return true; }
}
```

Testing without Indirect Inputs. Production code

```
var firstDeep = new FirstDeep(new SecondDeep());
```

Testing using a Test Stub

```
var firstDeep = new FirstDeep(new SecondDeepStub());
```

Test Doubles. Test Spy

- an object that acts as an observation point for the Indirect Outputs of the SUT
- quietly records SUT method calls
- useful for a method calls assertions

Test Doubles. Test Spy.Example

RemoveFlight() calls LogMessage()

```
[Test]
public void TestMethod(){
    //Arrange

    //fixture setup
    var expectedFlightDto = CreateAnUnregFlight();
    var facade = new FlightManagementFacadeImpl();

    //test double setup
    var logSpy = new AuditLogSpy();
    facade.SetAuditLog(logSpy).

    //Act
    facade.RemoveFlight(expectedFlightDto.GetFlightNumber());

    //Assert
    Assert.AreEqual(logSpy.GetNumberOfCalls(), 1);
    Assert.AreEqual(logSpy.GetDate(), helper.GetTodayDate());
    Assert.AreEqual(logSpy.GetDetail(), expectedFlightDto.GetFlightNumber());

}
```

Test Doubles. Mock Obj

- looks like a Test Spy
- contains predefined expectations
- 2 types:
  1. Strict – verify the expected calls are made in the exact predefined order
  2. Lenient – tolerates out-of-order calls

BEHAVIOR verification – verify if SUT **called** the exact expected methods

Test Doubles. Mock Obj. Example

RemoveFlight() calls LogMessage()

```
[Test]
public void TestMethod(){
    //Arrange

    //fixture setup
    var expectedFlightDto = CreateAnUnregFlight();

    //mock object setup/config

    var mockLog = ConfigurableMockAuditLog();
    mockLog.SetExpectedLogMessage(
                    helper.GetTodayDate(),
                    expectedFlightDto.GetFlightNumber());
    mockLog.SetExpectedNumberCalls(1);

    //mock instalation

    var facade = new FlightManagementFacadeImpl();
    facade.SetAuditLog(logSpy).
    //Act
    facade.RemoveFlight(expectedFlightDto.GetFlightNumber());

    //Assert
    mockLog.verify();

}
```

Test Doubles. Fake Obj

TO BE CONTINUED…

TEMA

Test Doubles. How to implement?

C# Code examples

- Dummy Object
- Test Stub
- Test Spy (Internal & Dependency)
- Mock Object
- Fake Object

Test Doubles. Dependency injection

Production code

```
public void TransferFundsFromEurAmount(Account destination,
float amountInEur)
        {
            var convertor = new CurrencyConvertor();
            var amountInRon = convertor.EurToRon(amountInEur);

            destination.Deposit(amountInRon);
            Withdraw(amountInRon);
        }
```

How do we domain test this SUT?

How do we test this SUT for different Eur/Ron rates?

Is this code designed for testability?

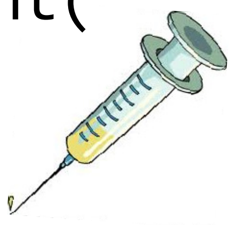
We inject the DOC (CurrencyConverter) into the SUT

Test Doubles. Dependency injection

A testable version of the production code

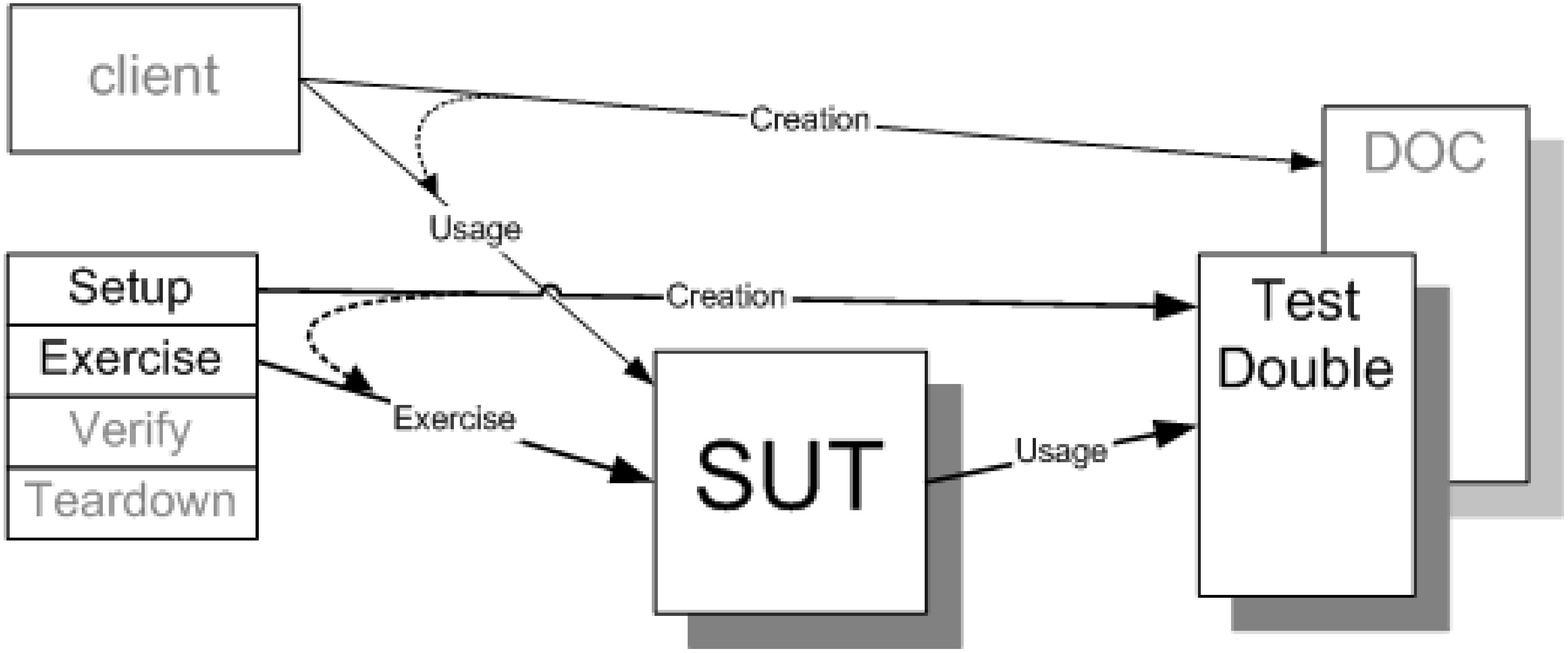
```
public void TransferFundsFromEurAmount(
     Account destination,
     float amountInEur,
     ICurrencyConvertor convertor)
{
    var amountInRon = convertor.EurToRon(amountInEur);

    destination.Deposit(amountInRon);
    Withdraw(amountInRon);
}
```



Test Doubles. Dependency injection

The client provides the DOC to the SUT

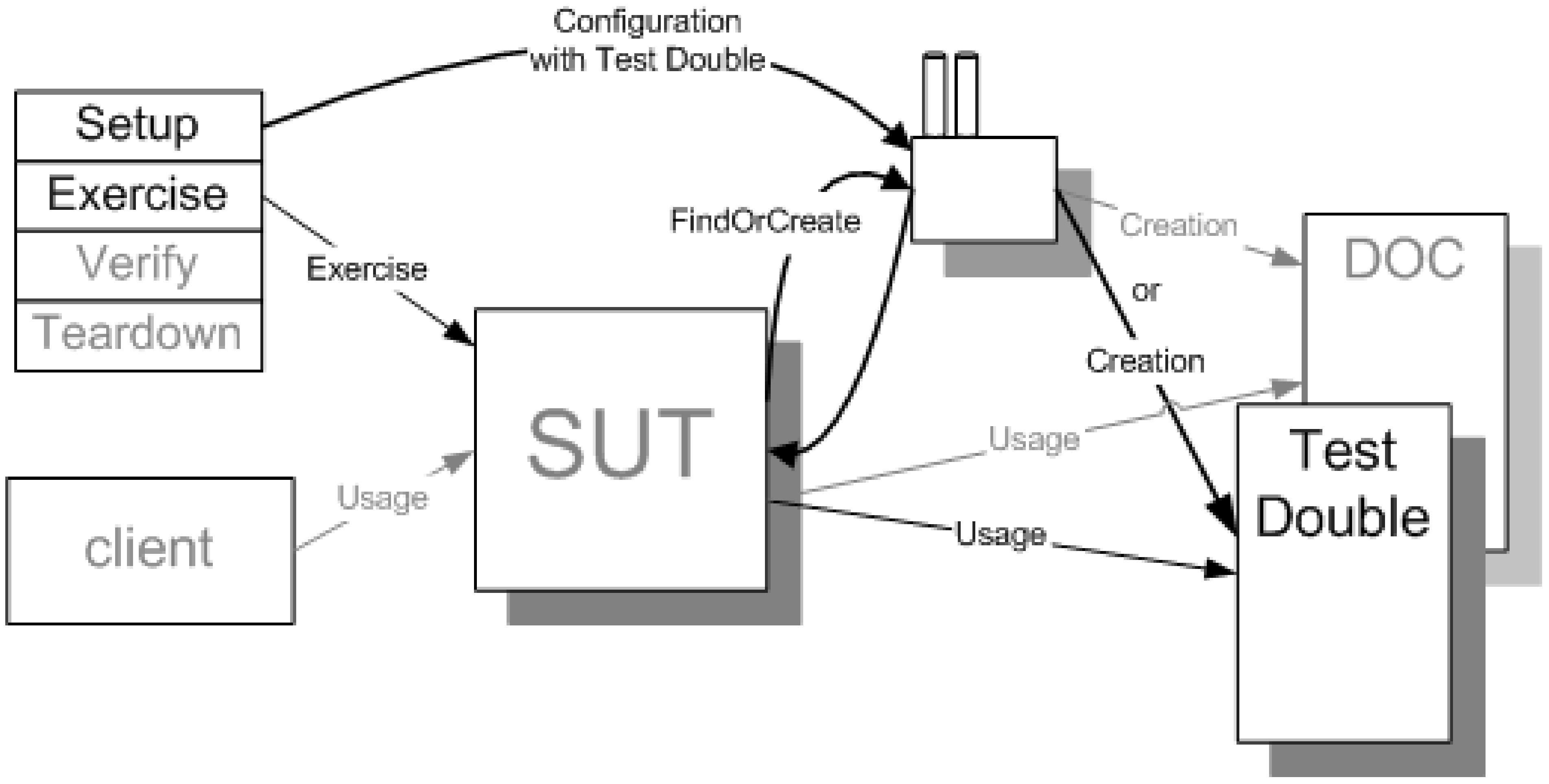


Test Doubles. Dependency injection types

- Constructor Injection
- Setter Injection
- Parameter Injection

Test Doubles. Dependency lookup

The SUT asks another object to return the DOC before using it.



Test Doubles. Dependency lookup types

- Object Factory

  the TestMethod tells (set) the ***Object Factory*** to ***create*** a TestDouble instead of DOC whenever a specific method is called

- Service Locator

  the TestMethod configures the ***Service Locator*** to ***return*** the TestDouble instead of DOC whenever the SUT requests it

Mocking Frameworks (C#)

- NSubstitute

  http://nsubstitute.github.io

- Moq

  https://github.com/Moq/moq4/wiki/Quickstart

- FakeItEasy

  https://github.com/FakeItEasy/FakeItEasy/wiki